(12)

EUROPEAN PATENT APPLICATION

(72) Inventor : **Arndt, Richard Louis
1607 Barnswallow Drive
Austin, Texas 78746 (US)**

Inventor : **Nicholson, James Otto
5003 Sevan Cove
Austin, Texas 78731 (US)**
Inventor : **Silha, Edward John
11509 Pyreneese
Austin, Texas 78759 (US)**
Inventor : **Thurber, Steven Mark
8308 Ephraim Road
Austin, Texas 78717 (US)**
Inventor : **Youngs, Amy May
10638 Floral Park Drive
Austin, Texas 78759 (US)**

(74) Representative : **Litherland, David Peter
I B M (UK) Ltd
UK Intellectual Property Department
Hursley Park
Winchester, Hampshire SO21 2JN (GB)**

(54) Scalable system interrupt structure for a multiprocessing system.

(57)   An interrupt subsystem within a data processing system is scalable from low-end uniprocessor systems to high-end multi-processor (MP) systems. This interrupt subsystem provides for queuing of interrupts from many sources, and for queuing of interrupts to the best processor in a multi-processor system. The external interrupt mechanism is separated into two layers, an interrupt routing layer and an interrupt presentation layer. The interrupt routing layer routes the interrupt conditions to the appropriate instance of an interrupt management area within the interrupt presentation layer. The interrupt presentation layer communicates the interrupt source to the system software which is to service/process the interrupt. By providing two layers within the interrupt subsystem, application or system software can be written which is independent from the types or sources of interrupts. The interrupt routing layer hides the details of a particular hardware implementation from the software. The interrupt presentation layer interfaces to the system and/or application software, and provides hardware independent functionality.
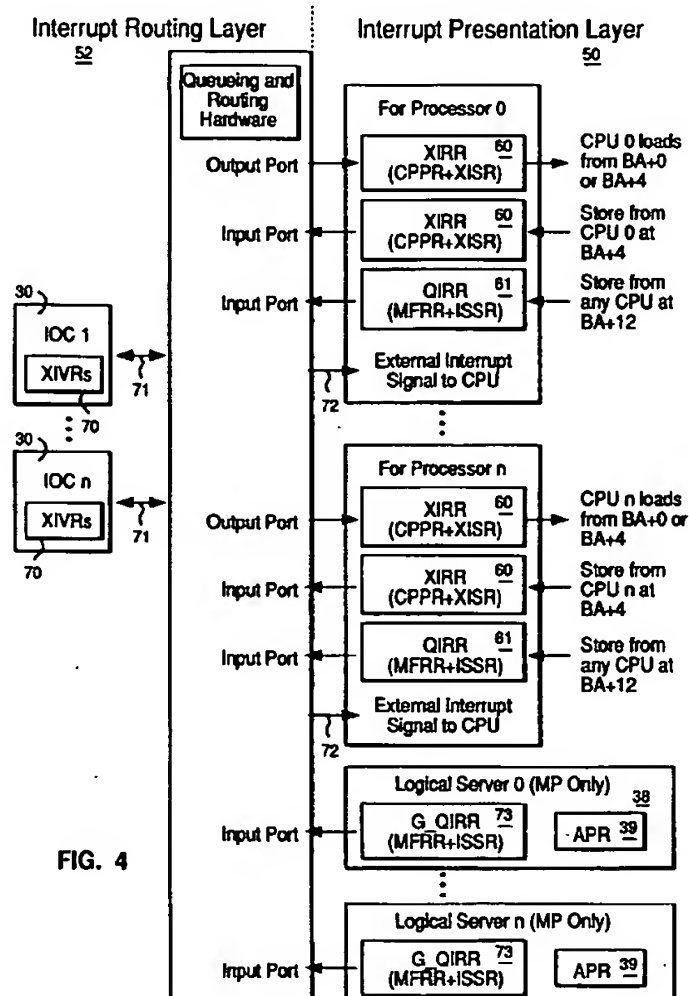
FIG. 4

EP 0 644 487 A2

This invention relates to data processing systems, and more particularly to a way of signalling interrupt information between an interrupt source and an interrupt processor in a data processing system.

In data processing systems, interrupts are used to signal a processor that an interrupt condition exists at a given source. This interrupt source could be, for example, an adapter card on a system bus which requires
5   some type of service. The required service may be to initiate a transfer of data, or to read a status register that has recently changed.

When the processor has been conditioned to accept an interrupt, otherwise known as having the interrupts enabled, the processor will initiate interrupt processing upon the receipt of an interrupt. This interrupt process- ing typically involves the processor interrogating the source of the interrupt, performing functions based upon
10   the type of interrupt, and resetting/turning off the interrupt.

Interrupt priorities have also been accommodated in traditional systems. If more than one interrupt signal is active at a given time, the use of interrupt priorities tells the processor which interrupt to service first.

Interrupt controllers have been designed to offload certain interrupt handshake functions that are required, such as resetting an interrupt signal. Typical of such interrupt controllers is an Intel 8259 controller, which is
15   described in the Intel Component Data Catalog, 1981 (available from Intel Corp. Literature Department, 3065 Bowers Avenue, Santa Clara, CA), and hereby incorporated by reference as background material. These in- terrupt controllers can monitor multiple interrupt sources, and only interrupt the processor using a single in- terrupt line.

Current interrupt signalling methods were primarily designed for uniprocessor systems with few interrupt
20   sources or priority levels. Most systems send interrupts over one or more interrupt lines hardwired on the pla- nar. In a multiprocessor environment, where there is more than one processor capable of servicing an interrupt, these types of interrupt signalling techniques cause increases in bus complexity. An interrupt signal from each interrupt source would have to be wired to each processor or interrupt controller capable of servicing such in- terrupt.

25   Attempts to satisfy needs of a multiprocessing data processing system have required dedicated interrupt controllers for each processor in the system. This approach is not only costly, but also does not allow for co- hesive management of interrupts which are generated from a plurality of sources that must be routed to a plur- ality of processors for servicing.

A method is needed that is expandable, allowing many interrupt sources and priority levels. In addition, a
30   method of signalling interrupt information is needed for use with multiprocessing systems that handle multiple processors and multiple interrupt controllers.

Current interrupt systems are hardware specific, with varying amounts of software dependencies. A change in the number or types of interrupts requires changes be made to the particular operating system soft- ware. An interrupt subsystem is needed that provides software independence from the underlying hardware
35   interrupt structure.

The present invention provides, in a first aspect a system for maintaining a plurality of interrupts in a data processing system having a plurality of processors, comprising: means for routing said at least one of said plur- ality of interrupts to a particular processor; and means for queuing said at least one routed interrupt to said particular processor.

40   Viewed from a second aspect, the present invention provides a system for processing a plurality of inter- rupts in a multi-processor data processing system, comprising: an interrupt routing layer for receiving at least one interrupt, said routing layer having means for routing said at least one interrupt to a particular processor; and an interrupt presentation layer having means for receiving said at least one routed interrupt from said in- terrupt routing layer and means for queuing said at least one routed interrupt.

45   The interrupt subsystem in which the present invention is preferably embodied is scalable from low-end uniprocessor systems to high-end multi-processor (MP) systems. This interrupt subsystem provides for queu- ing of interrupts from many sources, and for queuing of interrupts to the best processor in an MP system. This is achieved by separating the external interrupt mechanism into two layers, an interrupt routing layer and an interrupt presentation layer. The interrupt routing layer routes the interrupt conditions to the appropriate in-
50   stance of an interrupt management area within the interrupt presentation layer. The interrupt routing layer is scalable to support both low-end/low-cost and high-end/high-performance systems. The interrupt presentation layer communicates the interrupt source to the system software which is to service/process the interrupt. This software accepts the interrupt condition and is responsible for resetting the interrupt condition. This software also indicates the acceptance of the interrupt and notifies the interrupt presentation layer that it has processed
55   the interrupt.

By providing two layers within the interrupt subsystem, application or system software can be written which is independent from the types or sources of interrupts and from the underlying system hardware structure. The interrupt routing layer hides the details of a particular hardware implementation from the software. The

2

interrupt presentation layer interfaces to the system and/or application software, and provides hardware independent functionality.

In a preferred embodiment, an interrupt packet protocol is defined for both interrupt requests and interrupt resets. Address bus packets are used, and have many advantages over the hardwired method. When an interrupt source (hereinafter called an I/O controller, or IOC) has an interrupt request, the IOC requests the address bus. When granted the bus, the IOC sends an interrupt request packet across the address bus. The interrupt controller processes the information and signals an interrupt to the processor. In a multiprocessing system, the interrupt controller may route the interrupt information to an appropriate processor. When the processor is done processing the interrupt, it will alert the interrupt controller to send a reset packet to the IOC. The interrupt controller then requests the address bus. When granted the address bus, the interrupt controller sends an interrupt reset packet across the address bus.

Since there are usually multiple cycles of data transfer for each address, the data bus is more heavily used than address bus. By sending interrupt information over the address bus, it is possible to use an underutilized resource, the address bus, and not use the busier data bus. Another advantage of using the address bus is that, in the preferred embodiment, all the IC chips on the system bus use the entire address bus. However, these IC chips do not all use the entire data bus. Thus, the interrupt packets going over the address bus use the address lines already in the system at a cost of zero pins and zero wires. This is in contrast to prior methods, which use several distinct interrupt lines.

Since memory mapped I/O is the most widely used method of performing input and output operations, it is used as the method for sending interrupt packets. Being the most common method of performing I/O, all the chips on the system bus already have logic to do memory-mapped I/O.

In the preferred embodiment, the interrupt packet definition allows for 16 interrupt sources from each of 512 IOCs. There are 256 interrupt priority levels. Each system may have 256 interrupt servers. This allows the packet definition to be used in a wide range of machines. There are reserved bits in both the Interrupt Request Packet and Interrupt Reset Packet that may be used by systems that need to transmit more information in the interrupt packets. This packet definition thus allows for providing extendibility for future systems.

Older hard-wired interrupt systems provided little immediate information about an interrupt. The bus packet method provides interrupt source, IOID, priority, and server information all at the same time and in the same place. In addition to system flexibility, this makes system interrupt debug much easier. Information can be put into the reserved address bits 0–2 of the Interrupt Reset Packet to tell the IOC which kind of Interrupt Reset is being sent.

Since interrupts are transferred over the address bus, they are sequenced with other interrupts and with other system operations. This is helpful during debug because it makes interrupts more observable at a system level and less asynchronous to the system.

Since interrupt information is defined in various fields in the interrupt packet instead of being hard-wired on the planar, this interrupt system is very programmable. Interrupt priority, server number, interrupt source and I/O controller ID are all programmable by using this packet structure. Thus, the interrupt structure is more flexible than those of hardwired systems.

A preferred embodiment of the invention will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1 shows a block diagram of a data processing system.

Figure 2 shows a block diagram of a device interface to a data processor through an interrupt controller.

Figure 3 shows the components of a logical server interfacing to software queues.

Figure 4 shows the internal structure of an interrupt controller.

Figure 5 shows the hardware and software queue structure.

Figure 6 shows a flowchart of the overall interrupt subsystem, including the I/O controller, the interrupt routing layer, and the interrupt presentation layer.

Figure 7 shows a bus timing diagram for an interrupt request.

Figure 8 shows a bus timing diagram for an interrupt reset.

Figure 1 illustrates the logical view of a data processing system 20. The system interconnect 22 allows for the transfer of data among the various components of the system: the processor 24, memory 26, and I/O 28 attached via an Input/Output Controller (IOC) 30 which is directly attached to the system interconnect 22. There may also be optional system bus(es) 29 from the processor to which additional I/O and IOCs are attached. The system interconnect 22 may be one of several different constructs (for example, a system bus, a switch, etc.) and is system dependent. In the preferred embodiment, the system interconnect is a system bus.

The external interrupt structure for today's systems is required to span a wide range of system requirements; from the simple single user personal computer to multi-user systems consisting of a hierarchy of multiple processors. Previous techniques can not effectively address such a range. The programming interface

to, and the logical view of, the interrupt system needs to be consistent. A typical interrupt structure is shown in Figure 2, wherein a device 32 (comprising I/O 28 and IOC 30 of Figure 1) capable of generating an interrupt is interfaced to a servicing processor 24 through an interrupt controller 34. This invention identifies an improved interrupt control structure which is scalable across a wide range of systems yet keeps a consistent programming model.

Referring now to Figure 3, the logical view of the system as seen from the system software is that of n (up to 256) queues of events. Within each software queue 42 and 43 there exists a prioritized list of events. These events comprise hardware generated interrupts (e.g. external interrupts from the IOCs) and software generated interrupts (e.g. inter-processor interrupts). Each queue 42, 43 is associated with a logical server. Queues 0 through m are associated with logical server #0, and queues m+1 through n are associated with logical server #1. Additional logical servers having additional processors could similarly be configured. In a single processor system, there is one server and, therefore, one queue. In a multi-processor system, there would be one queue associated with each processor and at least one global queue associated with the collection of processors viewed as one logical server. For example, Figure 3 shows each queue 42 associated with a processor 40, and a global queue 44 associated with the collection of processors 0 through m. The collection of processors 40 is viewed as logical server #0 at 38. Similarly, each queue 43 is associated with a processor 41, and a global queue 45 is associated with the collection of processors m+1 through n. The collection of processors 41 is viewed as logical server #1 at 39. For multi-processor systems, there must be a mechanism for determining which processors are available for use by each global queue. This mechanism is provided by the Available Processor Register (APR) (to be further described with reference to Figure 4).

Continuing with Figure 3, associated with each queue 42, 43 is a server number in the range of 0x00 through 0xff. The individual processors 40, 41 within the complex are assigned server numbers ascending from 0x00, and queues which serve multiple processors (i.e. the global queues) are assigned server numbers descending from 0xff. Queue lengths (i.e. the length of the prioritized list within the queue) are implementation dependent, but have a minimum depth of one.

External interrupts are sourced from IOC(s), other processors in the complex, and from other sources in the system (for example, an emergency power off warning interrupt). While the different sources require different physical signalling mechanisms, the logical appearance to the server (either the logical server for the global queues or the processor for other queues) is as one queue headed by the most favoured event, as will be further described below. The highest priority (i.e. most favoured) interrupt is defined to be 0x00 and the lowest priority interrupt (i.e. least favoured) is defined to be 0xff. Hence, interrupt level 0x55 is more favoured than interrupt level 0xff and less favoured than interrupt level 0x00.

This invention separates the external interrupt mechanism (i.e. the interrupt controller) into two layers: (i) the interrupt presentation layer and (ii) the interrupt routing layer. These layers are shown in Figures 4 and 6. Referring to Figure 4, the interrupt presentation layer 50 comprises registers associated with processors or servers. The operating system software interfaces to these processors or servers to create and handle individual interrupts. The interrupt presentation layer 50 has a definition which only varies by the number of processors or servers within a system. The interrupt routing layer 52 routes the interrupts from the sources to the destinations and is by its nature far more implementation specific. System software may have to set up the configuration of the interrupt routing layer 52 at power-on time, but does not have to interface to this interrupt routing layer on a per interrupt basis.

The registers used to manage interrupts are now briefly described, and will be later described in more detail.

* Available Processor Register (APR)
    - Optional register to be used by the interrupt routing layer hardware to know which processors are available for a particular server for which to route interrupts. The APR(s) are intended to be used with global queues, and are shown at 39 of Figure 4.
* Global Queued Interrupt Request Register (G_QIRR)
    - This register may be written as a single byte or as four bytes (a 32 bit word), and is shown at 73 of Figure 4.
    - Used in Symmetric MP (SMP) systems as a non-processor specific server queue.
    - Consists of two registers:
        * Global Most Favoured Request Register (G_MFRR), which is an MFRR.
        * Interrupt Source Specification Register (ISSR)
            - Used in SMP systems to configure the source specification of a G_MFRR interrupt (see XISR).
* Queued Interrupt Request Register (QIRR)
    - This register may be written as a single byte or as four bytes (a 32 bit word), and is shown at 61 of Figure 4.

4

- At least one per processor plus one per non-processor specific server queue
- Consists of two registers:
  - Most Favoured Request Register (MFRR)
    - Holds the priority of the most favoured request on the queue
    - This register can be read back by the software to verify that the write has taken place.
  - Interrupt Source Specification Register (ISSR)
- External Interrupt Request Register (XIRR)
  - One per processor, shown at 60 of Figure 4, and provides a single source identifier to the system software for interrupts.
  - Consists of two registers:
    - Current Processor Priority Register (CPPR)
      - This register is updated by the software to contain the current processor priority
      - This register is updated, when the software issues a Load instruction to the XIRR at a certain address, to the priority of the interrupt represented by the XISR data which is delivered by the executed Load instruction.
    - External Interrupt Source Register (XISR)
      - Indicates the source of a pending interrupt (or a value of 0, if no pending interrupt)
- External Interrupt Vector Register (XIVR)
  - One per interrupt level in each IOC, shown at 70 of Figure 4.
  - Used to prioritize each interrupt level
  - Can be used to vector particular interrupts to particular servers

Interrupt Routing Layer

The goal of the interrupt routing layer is to direct the most favoured interrupt request to the processor operating at the least favoured priority, i.e. the best processor. To the best of its ability, the interrupt routing layer avoids sending an interrupt to a processor which is running at a more favoured level than the incoming request. This invention allows the interrupt routing layer to be implemented differently depending on the system requirements. Therefore, different implementations may approximate the above goal to different levels of accuracy and this will appear as a variable delay in the routing of interrupt requests. The greater the expected system load due to interrupts, the closer the interrupt routing hardware should approach the goal in order to achieve proper system performance. To fully achieve this goal, the interrupt routing hardware would have to be fully aware of the state of the system (that is, the exact per cycle processor priority, and the contents of all logical interrupt request queues). In practice this may not be possible because: (i) there may be more potential interrupt requests than the hardware queues within the interrupt routing layer; or (ii) the processor priority may take several cycles to propagate from the processor to the interrupt routing hardware, causing a certain level of uncertainty in the interrupt routing layer as to the exact per-cycle processor priority. The depth of queue problem is handled by requiring the IOCs to resubmit interrupt request messages that are rejected by the interrupt routing hardware. This allows the interrupt routing hardware to implement a queue depth (minimum of one) to satisfy some percentage of the expected cases, with the interrupt rejection mechanism used to handle any overflow cases. The system software is unaware of the rejection mechanism which only exhibits a variable latency to affected interrupt requests. Since the interrupt routing hardware may be unaware of the true processor priority when it first routes a request toward a processor, the routing hardware must be prepared for the software to change its operating priority after the interrupt routing hardware has initially assigned a request to a specific processor. Failure to take changing processor priority into account can result in priority inversions and severe system performance degradation (priority inversion can occur if an interrupt is queued which has a less favoured priority than the current processor priority, and if that queued interrupt prevents an interrupt with a more favoured priority than the processor from getting in and interrupting the processor); the queued interrupt will not get serviced until the processor drops down in priority below the queued interrupt priority. Again, the interrupt rejection mechanism may be used to recover from queue resource problems. By rejecting the interrupt back to the IOC, the IOC acts as an extension to the interrupt routing layer queuing mechanism.

An interrupt source will typically send a particular interrupt to the interrupt routing layer via interface 71 (Figure 4) only once, unless one of the following occurs: (i) the interrupt is rejected by the interrupt routing layer; or (ii) the interrupt is reset by the software writing to the XIRR with an XISR value equal to the interrupt source and the interrupt for the interrupting source is still pending (i.e. has not yet been serviced). This will be further described below with reference to Figure 6.

Various implementations of the interrupt routing layer 52 are possible, including (i) a single element queue per processor consisting of an external interrupt priority and source for low-end machines, or (ii) multiple ex-

ternal interrupt source queue registers for high-end machines. The hardware assures that the exact hardware queuing implementation within the interrupt routing layer is transparent to the system software which interfaces to the interrupt presentation layer. There are numerous types of queuing techniques known in the art that may be used, such as a hardware register file whose entities are kept in priority order, described in US patent 4,807,111 are hereby incorporated by reference. A key feature of the present invention is the use of such queues by an interrupt presentation layer. This interrupt presentation layer provides software transparency from the underlying queue implementation within the interrupt routing layer.

At system setup time, the setup software must determine the Interrupt Routing Layer configuration, including how many logical servers are supported, which processors support which logical server, and which interrupts are to be directed to which server. This is done by reading some implementation specific registers, or by having the configuration information stored in ROM. APR's are used to specify which processors work against which logical server queue. The specific setup determination is hardware dependant, and will vary between differing hardware implementations. The only requirement is that the above listed setup information, however determined, is placed in the APRs for subsequent access by the interrupt routing layer.

Interrupt Presentation Layer

. The inter-processor interrupt mechanism has a physical queue of request blocks (57 of Figure 5) in system memory that are associated with each queue 42, 43 of Figure 3. Each of these software-managed queues are maintained in priority order by the software. The implementation of the queue is not defined in the interrupt mechanism, but rather is left up to the operating system software. The implementation of the various queues may be different depending upon the expected frequency of use. Associated with each queue is a Most Favoured Request Register (MFRR) (in system memory space). When a program needs a service performed by a particular processor, it enqueues a request block to that processor's queue, it determines if the new request is at a more favoured priority, and if it is, the value of the new request's priority is written into the MFRR. When a program dequeues a request for service, the priority value of the next request on the queue is loaded into the MFRR. If the queue is empty after dequeue, the least favoured value (0xff) is loaded into the MFRR. A value other than 0xff in the MFRR indicates to the interrupt hardware in the interrupt routing layer that an interrupt of that priority should be signalled at 72 to a (the) processor which services that queue.

Each processor has associated with it a memory mapped Interrupt Management Area which contains the external Interrupt Request Register (XIRR) 60. The XIRR is a 4-byte facility and is composed of two fields: the Current Processor Priority Register (CPPR) and the External Interrupt Source Register (XISR).

The CPPR contains the processor's operating priority. The CPPR may be written by the system software to prevent interruptions from less favoured priority requests. The interrupt routing layer will only direct an interrupt request to a processor if its CPPR field is less favoured than the priority of the interrupt request. The system software stores to the CPPR as a 1-byte register to inform the interrupt hardware in the interrupt routing layer of the current operating priority of the processor.

To determine the source of the interrupt, the system software reads the XISR by issuing a Load instruction to the XIRR. The value in the XISR specifies the source of the interrupt (if an IOC, which IOID and level; if a processor, which server queue). Based upon this information, the software can determine the proper program or process to invoke to service the interrupt. The XISR presents the appearance of a read only register from the interrupt routing layer for Load operations, and a write only register to the interrupt routing layer for Store operations. That is, what is written is not automatically what is read, since the hardware interprets a write to the XISR to be a signal to send an interrupt reset to the interrupting source (and not a signal to directly write to the XISR bits themselves, as will be later described). The XISR must be accessed atomically along with the CPPR when a 4-byte accesses is directed to the XIRR. For interrupts from IOCs, the upper bits of the XISR indicate the IOC IOID, and the low order 4 bits of the XISR field define up to 16 sources (or levels) within an IOC. Several values of this register have defined meanings. The XISR will be further described below.

The interrupt presentation layer of the interrupt mechanism is embodied through an Interrupt Management Area for each processor in the system, as shown below in Table 3. The Interrupt Management Area is within the data processing system's memory space. The starting address of the processor's interrupt management area is referred to as its Base Address (BA) for the rest of this document. The BA is different for each processor (that is, there is a separate Interrupt Management Area for each processor), and these areas can be accessed by any processor (there is no protection against it). The BA for a processor is setup at configuration time. The layout of the Interrupt Management Area is as follows:

Address	Byte 0	Byte 1	Byte 2	Byte 3	Comments
BA+0	CPPR		XISR		XIRR without side effects
BA+4	CPPR		XISR		XIRR with Load/Store side effects
BA+8	DSIER		DSIER		Data Storage Interrupt Error Register
BA+12	MFRR		ISSR		Required QIRR
BA+16	MFRR		ISSR		Optional 2nd QIRR
BA+20	MFRR		ISSR		Optional 3rd QIRR
000	MFRR		ISSR		Optional nth QIRR
TABLE 1: Interrupt Management Area: Interrupt Presentation Layer Registers					

Queuing

Figure 5 shows the interrelationship between the various queues previously described. The software queue 42, 43 that provides the consistent, hardware independent interface to the system software is shown for a particular processor X at 53. Queue X interfaces to processor X through the XIRR register shown at 54. This XIRR register can be modified by the interrupt routing layer 52 at 55. The step of selecting the highest priority by the routing layer at 55 must account for a plurality of interrupt types. Hardware interrupts which are maintained in a hardware queue at 56 are presented to the interrupt routing layer 52 at 58. These hardware interrupts are sourced from individual IOC hardware queues at 51. In addition, software generated interrupts may be presented to the interrupt routing layer 52. These software interrupts (e.g. inter-processor interrupts) are maintained in a software managed queue at 57, and presented to the interrupt routing layer at 59 through the MFRR register.

In summary, the software queues 42 and 43 of Figure 4 which are presented to the system software are a combination of software-managed queues and hardware interrupt queues. Thus, the hardware queues can be distributed between the routing layer and the IOCs. Further, the hardware-generated and software-managed queues are presented to the system software using a uniform interface, the XIRR register.

Interrupt Handling

The overall flow of the interrupt handling is shown in Figure 6. When an IOC has an interrupt which needs servicing at 74, it first determines if this is the highest priority which needs to be presented. If there is a higher priority interrupt that has not been serviced yet, it will present that interrupt first at 75. The IOC will select the XIVR associated with the particular interrupt to be presented and will send the Server number and priority from the XIVR, along with the IOC IOID and a number indicating the source within the IOC which is requesting service to the interrupt routing layer at 76. The interrupt routing layer can use the server number from the interrupt information to direct at 77 the interrupt to a specific processor (server) 98 or if the server number does not correspond to a particular processor in the system, to choose which processor to which to route the interrupt at 78. How the hardware chooses to which processor to route the interrupt in this latter case is implementation dependent, and can be optimized based on the system design point (for example cost on the low end or performance on the high end). Information which comes into this routing decision block 78 is what priority the various processors are operating (the CPPR value) and what other priorities are queued up waiting for that processor (for example, the MFRR or G_MFRR for that processor). If the priority of the interrupt received from the IOC is less favoured or equal priority to existing interrupts queued at 79, then the interrupt is rejected at 96 back to the IOC, and the IOC must represent the interrupt at a later time (for example, after some fixed amount of time has elapsed). If the interrupt from the IOC is more favoured than an existing queued interrupt which has already been placed into the XIRR of the target processor 80, then the interrupt placed into the XIRR previously is rejected back to the IOC from which it came at 95, and the new interrupt takes its place at 81. Although a particular implementation of an interrupt routing layer mechanism may chose to queue the interrupt in the routing layer for presentation at a later time, in the preferred embodiment the routing layer rejects the interrupt back to the source, and the source represents the interrupt at a later time. The loading of the XISR must be atomic; that is, the hardware must guarantee that the system software and interrupt routing layer are not trying

7

to access the XISR simultaneously (for example, if the hardware is trying to update the XIRR with a higher priority request at the same time the processor was trying to read the XIRR).

When the XISR contains a non-zero value at 83, an interrupt is signalled to the processor by the hardware via the interrupt signal into the processor hardware. When the interrupt is signalled and when the interrupts are enabled at 84, the software receives the interrupt signal and the interrupt processing begins at 85. The software, at the beginning of the interrupt processing, reads the XIRR at 86 and saves that value away until the end of the interrupt processing. When the software reads the XIRR, the hardware will place the priority of the interrupt represented by the XISR value into the CPPR and set the XISR to 0 at 86. Setting of the CPPR to the value of the priority of the interrupt prevents the software from having to do that; the processor priority at the start of service of that interrupt will be equal to the priority of the interrupt, so that the interrupt routing layer will not interrupt that servicing with a less favoured interrupt. By setting the XISR to 0, the interrupt signal to the processor is deactivated, and, if the processor would read the XIRR later, a value of 0 in the XISR would signal that no interrupts are pending. Once an interrupt has been presented to a processor by the processor reading the XIRR at BA+4, this signals the hardware that the software will start processing the interrupt, and therefore, the interrupt routing layer may neither preempt nor cancel the request. After reading the XIRR, the software uses the value in the XISR field of the XIRR to determine what interrupt servicing routine to invoke. If the XISR points to the QIRR at 87, the software removes the most favoured queue entry from that queue and sets the MFRR in the QIRR to the value of the priority of the new most favoured entry on that queue 88 or to the value of 0xff, if the queue is then empty. If the XISR points to the G_QIRR at 89, the software removes the most favoured queue entry from the global queue and sets the G_MFRR in the G_QIRR to the value of the priority of the new most favoured entry on that queue at 90, or to the value of 0xff if the queue is then empty. If the XISR doesn't point to one of the software queues, then it points to an external interrupt service routine, and the software invokes the appropriate device driver to service the interrupt at 91. In the case of an external interrupt, after the device driver has serviced the interrupt, it will reset the interrupt in the IOC during the course of the service routine (for example, most hardware would reset the interrupt in the IOC by issuing a store instruction to a certain address in the IOC's address space). No matter what the source of the interrupt, at the end of the servicing, the software will write the XIRR at BA+4 with the value which was read from the XISR and saved at the start of the interrupt servicing 92. When the XIRR is written at BA+4, the CPPR will be set to the value in the store data (which, in this case, will be the value of the CPPR at the beginning of the interrupt servicing) and will issue an interrupt reset to the interrupt routing layer at 93 and IOC at 94 with the IOID and source within the IOC as specified in the XISR data that is written (which, in this case, will be the value of the XISR at the start of the interrupt, so this will reset the interrupt which has just been serviced). When the IOC receives the reset, if the hardware still thinks that an interrupt is pending at 97 (for example, if the interrupt has occurred again since the software servicing and the write to the XIRR), then the interrupt presentation process starts over again at 74.

The XIRR facility appears twice in the external interrupt management area. Address BA+0 is designed to be used with interrupt polling. Address BA+4 has side effects when read or written, and is designed to allow efficient interrupt handler software by having the hardware assist the software in the interrupt queuing process. The registers and their use are further described below.

Processor to Processor Interrupts

The Most Favoured Request Register (MFRR) holds the priority of the most favoured request queued on a software managed queue for this processor. When written to a value other than 0xff, the MFRR competes with other external interrupts for the right to interrupt the processor. When the MFRR's priority is the most favoured of all interrupt requests directed to the processor, an appropriate value is loaded into the XISR (see XISR register description), and an interrupt is signalled to the processor. When the processor reads the XIRR at BA+4, the value in the MFRR will be loaded by the hardware into the CPPR. The MFRR may be read back by the software to ensure that the MFRR write has been performed.

During the processing of an inter-processor interrupt, the highest priority request is dequeued by the software from the software queue associated with the MFRR and the priority of the next favoured request is loaded into the MFRR by the software, using traditional queue management techniques known in the art.

Global Queues

In MP systems, the system memory space contains one or more Global Queue MFRR's which are used by software to send inter-processor interrupts to any processor within some server group. The Global Queue MFRR's work just like the per processor MFRR described above except that the interrupt routing layer deter-

mines the processor to receive the request based upon its own algorithm, and the value loaded into the XISR is an IOID which indicates the Global Queue. This routing algorithm can similarly be optimized based upon the system design point. For example, a system optimized based on cost might route to a random processor to prevent from having to implement priority comparison logic, whereas a system which is implemented for high

5   performance would attempt to always route to the processor which is running at the least favoured priority.

External Interrupt Vector Register (XIVR)

Each IOC 30 contains one External Interrupt Vector register 70 for each external interrupt that it will sup-
10   port. The bits in each of these registers are defined as follows:

BITS	DESCRIPTION
0-15	Reserved: These bits are reserved and should be set to a value of 0 by the software on a Store instruction. These bits will be returned by the hardware as a value of a 0 on a Load instruction (software note: these bits are only guaranteed to be 0 as long as this field remains "reserved": if these bits are redefined in the future, software may get back something other than 0).
16-23	Interrupt Server Number: This determines where the interrupt is to be directed by the interrupt routing hardware. If the value represents a processor in the system, then the interrupt will be directed to that processor. If it does not correspond to a particular processor, then the interrupt routing layer can route based on its presentation algorithm.
24-31	Interrupt Priority: This field specifies what priority should be assigned to the incoming interrupt.

25   Table 2: XIVR REGISTER DESCRIPTION

These registers reside in the IOC address space, and the addresses of these registers are defined by the particular design of the IOC.

30   External Interrupt Request Register (XIRR)

The XIRR is a 4-byte register at addresses BA+0 and BA+4. Issuing a Load instruction to the XIRR at ad-
dress BA+0 causes the content of the CPPR and XISR to be sent to the processor with no side effects. This is designed for software polling of external interrupts. Issuing a Load instruction to the XIRR at address BA+4
35   has the following atomic side effects:
Prior to returning the content of the XIRR:
    * The interrupt signal 72 to the processor, whose BA is being accessed, is deactivated by the interrupt routing layer at 86 of Figure 6. The content of the XIRR is returned to the requesting processor only after enough time has elapsed to assure that the deactivated interrupt signal has propagated through the
40   processor's internal interrupt logic.
After the data from the XIRR is sent to the processor:
    * The contents of the CPPR are set to the priority of the interrupt signalled in the XISR (if XISR was zero, the CPPR field is unmodified)
    * The XISR is reset to 0x000000. Subsequent interrupt requests of more favoured priority will now cause
45   an interrupt to be signalled and presented in the XIRR.
When the system software begins to process an interrupt, the processor disables the interrupts -- masking off any subsequent external interrupts. During the interrupt processing sequence, the software must enable the interrupts to allow subsequent interrupts to be presented. Care must be taken to insure that the contents of the XIRR at BA+4 have been returned to the processor prior to enabling interrupts, in order to avoid a race
50   with interrupt routing layer's interrupt signalling termination. Such a race can produce undefined results. One way to insure the data has returned, in the face of potential processor speculative instruction execution, is to place an XIRR value data dependency in the code prior to the code which enables the interrupts. The processor register which is receiving the XIRR value is compared with itself, and a branch-if-equal instruction is executed using the results of this comparison, with the branch target address being the next instruction location.
55   This setting of the CPPR on the Load of the XIRR has the effect of rejecting all less favoured or equal priority external interrupts from the time that the software issues a Load instruction to the XIRR until it issues a Store instruction to the CPPR with a new value.
Issuing a Store instruction to the XIRR facility at BA+0 is undefined (data is ignored). Storing to the XIRR

9

at address BA+4 has atomic side effects and the effects are different for a 1-byte versus a 4-byte Store. When the Store instruction is a 1-byte Store, then this is a Store to the CPPR (see CPPR Register description). When the Store instruction is issued to the XIRR with a length of 4 bytes, an interrupt reset is sent to the source as indicated in the data that accompanies the Store to the XISR (not to the source indicated in the XISR at the time of the Store). The data that accompanies the Store to the XISR is not written into the XISR (and will not be obtained if subsequently read with a Load instruction). Rather, it is used to indicate the source to be reset to the interrupt routing layer (and to the IOC for interrupts other than inter-processor and global queue interrupts). Issuing a Store to the XISR at this address allows the source to present subsequent interrupts at the level indicated in the data accompanying the Store instruction. For a 4-byte Store, byte 0 is stored in the CPPR, but the system software ensures that this store of the CPPR is of a less favoured or equal priority than the previous CPPR value, because hardware is not required to handle dual resets for this case (one for the change in CPPR value to a more favoured or equal priority, and one for a write to the XISR).

If the interrupt routing layer hardware elects to reject interrupts on a CPPR change from a more favoured to a less favoured level, then on a 4-byte Store to the XIRR, the interrupt routing layer hardware can do one of two things:

1. Send two rejections back to the sources (one for the CPPR change and one for the XISR write)
2. Send the rejection for the XISR write (using the fact that the write of the CPPR must be of a less favoured priority than the previous CPPR value; a requirement placed on the software).

In order for the herein described interrupt mechanism to be used across differing systems potentially having different processors, the interrupt routing layer must insure that all interrupt signal constraints of a particular processor are met. If, for example, the processor does not internally latch the interrupt signal, and is not able to guarantee proper operation if the interrupt signal was deactivated during the processor's interrupt presentation cycle, then the interrupt routing layer should externally latch the interrupt signal resetting the latch with the read of the XISR rather than as the result of an interrupt rejection.

At the end of the interrupt handler, writing the XIRR at BA+4 with the value read at the start of the interrupt handler has the combined effects of issuing an explicit end of interrupt to the IOC and returning the processor's operating priority to its pre-interrupt value.

When software is polling interrupts, after the software decides to take an interrupt, the software must signal the hardware that the interrupt is being taken by issuing a Load to XIRR at BA+4. Software must then compare the XISR with the value it read at BA+0 to make sure that the hardware did not change the XISR between the Loads.

Current Processor Priority Register (CPPR)

This register is a 1-byte register and is used to contain the current priority of the processor to which it is associated. This register is at addresses BA+0 and BA+4. Issuing a 1-byte Load to the CPPR (at either BA+0 or BA+4) has no side effects. The CPPR is a field in the XIRR register. Each processor has its own CPPR. When the processor changes state, software may store the process priority of the processor into the CPPR. The interrupt routing layer rejects all interrupts for a processor that are at a priority less favoured to the CPPR priority. Thus, keeping the CPPR current will prevent an external interrupt from interrupting a process of more favoured or equal priority.

The CPPR is a field in the XIRR and can be read or written as part of the XIRR. When the contents of the CPPR are changed, the interrupt routing layer insures that only interrupts of more favoured level are signalled to the processor and are presented in the XIRR. This is accomplished using a hardware comparator. If the incoming interrupt priority is less than or equal to the current CPPR, then the interrupt is rejected back to the source. It is acceptable for the hardware to recompute (possibly by interrupt rejection) the most favoured interrupt to present after any Store to the CPPR if the direction of change of priority is to a less favoured priority, and it is mandatory if the Store to the CPPR is a more favoured or equal priority to any queued interrupt. When an interrupt is removed from the XIRR because of a store to the CPPR, if there is no interrupt of higher priority (higher than the new CPPR value) waiting to replace it in the XISR, then the interrupt presentation hardware sets the XISR to a value of 0 (atomically, with the CPPR store), indicating that no interrupt is pending, and lowers the interrupt request line to the processor.

External Interrupt Source Register (XISR)

This register is a 3-byte register and contains the address of the source of the interrupt. Each processor has its own XISR. This register is at address BA+1 and BA+5. However, this register must be accessed as part of the XIRR for purposes of atomicity. Issuing a Load instruction to the XIRR at address BA+0 causes the con-

tent of the XISR to be returned to the processor with no side effects. This is designed for software polling of external interrupts. Issuing a Load instruction to the XIRR at address BA+4 has the side effects of resetting the XISR to 0x000000 atomically after the contents of the XIRR have been returned to the processor. Subsequent interrupt requests of more favoured priority will then cause an interrupt to be signalled and presented in the XIRR.

Several values have special meanings in the XISR register, as described and shown below in Table 3.

Value	Meaning
0x000000	Reset: This value indicates that there are no current external interrupts pending. The XISR takes on this value as a side effect of a load from the XIRR at location BA+4.
0x000001	Early Power Off Warning (EPOW): This value indicates that an EPOW interrupt is pending.
0x000002 up to but not including the first IOID value	Inter-Processor Interrupt (IP): These value indicate that an Inter-Processor Interrupt is pending (see the definition of the MFRR, below). There is one value allocated per MFRR. In an MP System, each processor has an MFRR with its XISR value of 0x000002. The system configuration software will have to setup the first IOID in the system such that the value loaded into the XISR for the lowest IOID will be greater than the largest previously defined IOID values.

TABLE 3: XISR Special Values

Queued Interrupt Request Register (QIRR)

The Queued Interrupt Request Register is a 4 byte register with the first byte being the Most Favoured Request Register (MFRR) and the remaining low order 3 bytes as being the ISSR, if implemented. Software may write either a single byte, the MFRR, or a full four bytes.

Most Favoured Request Register (MFRR)

This is a 1-byte register. The content of this register is controlled by system software and indicates the most favoured interprocessor (IP) interrupt priority on an IP interrupt queue 42, 43 for the processor or server to which the particular MFRR is associated. If an MFRR for a processor is set to a value of 0xff, then there are no items in that IP interrupt queue 42, 43 for that processor, and the hardware is not to signal an IP interrupt to that processor. When the system software puts something on an IP queue 42, 43 for a processor, it will also set this register to the priority of the most favoured item on the IP queue. When this register is a value other than 0xff, it is treated by the interrupt routing layer as another interrupt request competing for presentation to the processor via the XIRR. When the value in an MFRR is the most favoured of all interrupt requests directed to the processor, an appropriate value is placed into the XISR (see XISR description) and an interrupt is signalled to the processor. As with all other interrupt sources, an MFRR interrupt shall be resubmitted if and only if the interrupt condition has not been reset when either the interrupt is rejected by the routing layer or software issues the end of interrupt. The interrupt condition is taken to be an MFRR value other than 0xff. Therefore, once the MFRR has a non-0xff value, and the interrupt routing layer has started to route the interrupt, the interrupt routing layer will not reroute the interrupt request to the interrupt presentation layer because of a subsequent change of value in the MFRR. The only way that the interrupt routing layer will reroute the MFRR interrupt request is due to an interrupt rejection or an interrupt reset (given that the MFRR does not have the value 0xff). The MFRR's value is only changed by a software store operation. Each processor has at least one MFRR. There is also one or more global MFRR's in an MP system. The MFRR's associated with a specific processor's IP interrupt mechanism is located at an address of BA+12, BA+16, and so on.

Global Queued Interrupt Request Register (G_QIRR)

The Queued Interrupt Request Register 73 is a 4 byte register with the first byte being the Global Most Favoured Request Register (G_MFRR), which is an MFRR, and the remaining low order 3 bytes as being the ISSR. Software may write either a single byte, the G_MFRR, or a full four bytes. The starting address for the global server's interrupt management area is referred to as its base address (BA). The BA for each global server is different, and is established at setup time. The layout of the global server interrupt management area is as follows:

Address	Byte 0	Byte 1	Byte 2	Byte 3	Comments
BA+12	G_MFRR		ISSR		Required QIRR for MP Systems
BA+16	G_MFRR		ISSR		Optional 2 nd QIRR
BA+20	G_MFRR		ISSR		Optional 3rd QIRR
000	G_MFRR		ISSR		Optional nth QIRR

TABLE 4: Interrupt Management Area: Interrupt Presentation Layer Registers

The ISSR (Interrupt Source Specification Register) contains the value to be loaded into the XISR when the interrupt associated with the corresponding MFRR is signalled to a processor.

Interrupt Signalling Between Device and Controller

Table 5 details the interrupt packet format designed for the IBM PowerPC 601 system bus. This system bus is further described in the "PowerPC 601 RISC Microprocessor User's Manual", 1992, and hereby incorporated by reference as background material. The same address bit definitions could similarly be used for other data processing machines using other processors. The Transfer Type bits are specific for the 601 bus and may change in other bus architectures.

OPERATION	TRANSFER TYPE	GLB	ADDRESS BITS 0-2	ADDRESS BITS 3-11	ADDRESS BITS 12-15	ADDRESS BITS 16-23	ADDRESS BITS 24-31
Interrupt Request	10110	1	RESERVED Default value B'000' Optional values below B'001' - First Presentation of interrupt B'010' - Representation of a rejected interrupt	IOID	Interrupt Source	Server Number	Interrupt Priority
Interrupt Reset	10111	1	RESERVED Default value B'000' Optional values below B'001' - Interrupt Reset caused by store to XIRR B'101' - Interrupt Reject due to store to CPPR B'110' - Interrupt Reject due to XISR being full B'111' - Interrupt Reject due to priority of request vs. CPPR	IOID	Interrupt Source	Server Number	RESERVED Set to X'00'

TABLE 5

The transfer type field is used to identity the type of transfer occurring on the address bus, e.g. an interrupt request or an interrupt reset. The global (GBL) bit is used to indicate that every device on the bus should monitor the transaction, and is always enabled for interrupt packet transfer. The address bit lines are grouped into 4 subsets. Address bits 0-2 are used to provide further information for the particular type of transfer which is occurring. For an interrupt request operation, address bits 0-2 indicate whether this a first or subsequent presentation of an interrupt, and thus indicates whether this interrupt was previously rejected and is being re-submitted. For an interrupt reset operation, address bits 0-2 indicate the cause of the interrupt reset. Address bits 3-11 specify the system address or route of the requesting IOC. Address bits 12-15 identify which one of up to 16 sources within the IOC made the request. Address bits 16-23, the server number, are taken directly from the XIVR register of the interrupt source. The server bits are compared, by each server, to the server number

13

written to a configuration register at set-up time for that particular server. Address bits 24–31 in the request packet contain the interrupt's priority and also come from the XIVR of the interrupt source. These priority bits are compared against the current value if the CPPR to determine if an interrupt request will be accepted. Address bits 24–31 in the reset packet are reserved, and contain X'00'.

5   Figures 7 and 8 depict the bus signalling of the data processing interrupt subsystem. Referring first to Figure 7, when an IOC has an interrupt request, the IOC requests the address bus from the central bus arbitrator by activating its bus request signal, BR 100. The arbitrator grants the bus by activating the corresponding bus grant (BG) signal 102 to the IOC. The IOC then sends an interrupt request packet over the address bus. This is done by driving the transfer start (TS) 104, bus address 110 and transfer-type (TT) 108 lines to certain values

10   (as defined in Table 5) to tell the interrupt controller there is an interrupt of a certain source and priority. In addition, signal address bus busy (ABB) 106 is driven low by the IOC to indicate that the address bus is in use. The interrupt controller processes the information and signals an interrupt to the processor. In a multiprocessing system, the interrupt controller may route the interrupt information to an appropriate processor, as previously described. The address acknowledgment signal (AACK) is driven low by the bus arbitrator to indicate

15   that the address portion of the transaction has been completed.

         Referring now to Figure 8, when the processor is done processing the interrupt, it will alert the interrupt controller to send a reset packet to the IOC (storing to XIRR register, as previously discussed). The interrupt controller then requests the address bus from the central bus arbitrator by activating its bus request signal, BR 100. The arbitrator grants the bus by activating the corresponding bus grant (BG) signal 102 to the IOC.

20   The IOC sends an interrupt reset packet over the address bus. It does this by driving the TS 104, bus address 110 and TT 108 lines to certain values (as defined in Table 5) to tell the IOC that the interrupt has been serviced. In addition, signal address bus busy (ABB) 106 is driven low by the IOC to indicate that the address bus is in use. The address acknowledgment signal (AACK) is driven low by the bus arbitrator to indicate that the address portion of the transaction has been completed.

25   A tenure consists of transferring one address and between 0 and 8 cycles of data. Since there are usually 4 cycles of data transfer for each address, the data bus is more heavily used than address bus. By sending interrupt information over the address bus, it is possible to use an underutilized resource, the address bus, and not use the busier data bus. Another advantage of using the address bus is that in the preferred embodiment, all IOCs currently on the system bus use the entire address bus. However, the IOCs do not all use the

30   entire data bus. The interrupt packets going over the address bus use the address lines already in the system at a cost of zero pins and zero wires. This is in contrast to prior methods, which used several distinct interrupt lines. Since memory mapped I/O is the most widely used method of I/O, it is used as the method for sending interrupt packets. Being the most common method of I/O, and using traditional techniques known in the art, all the chips on the system bus already have logic to do memory-mapped I/O.

35   In the preferred embodiment, the interrupt packet definition allows for 16 interrupt sources from each of 512 IOCs. There are 256 interrupt priority levels. Each system may have 256 interrupt servers. This allows the packet definition to be used in a wide range of machines. There are reserved bits in both the Interrupt Request Packet and Interrupt Reset Packet that may be used by systems that need to transmit more information in the interrupt packets. This packet definition thus allows for providing extendability for future systems.

40   Older hard-wired interrupt systems provided little immediate information about an interrupt. The bus packet method provides interrupt source, IOID, priority, and server information all at the same time and in the same place. In addition to system flexibility, this makes system interrupt debug much easier. Information can be put into the reserved address bits 0-2 of the Interrupt Reset Packet to tell the IOC which kind of Interrupt Reset is being sent.

45   Two hundred fifty six (256) possible interrupt servers are allowed in the interrupt packet definition. This means that up to two hundred fifty six (256) processors could be in the system receiving interrupts. This packet definition allows interrupts to be easily routed to different processors. In addition, the two hundred fifty six (256) possible interrupt servers can be defined for each system, to allow up to two hundred fifty six (256) interrupt controllers in a system. Each interrupt controller can service a single processor, or a batch of servers

50   can service each processor, as previously described.

         Since interrupts are transferred over the address bus, they are sequenced with other interrupts and with other system operations. This is helpful during debug because it makes interrupts more observable at a system level and less asynchronous to the system. For example, the system bus can be monitored to detect interrupts being presented on the address bus. Since the address packets are sequenced (i.e. not overlapped or con-

55   currently presented), it is easier to determine what actions invoked or otherwise caused subsequent actions.

         Since interrupt information is defined in various fields in the interrupt packet instead of being hard-wired on the planar, this interrupt system is very programmable. Interrupt priority, server number, interrupt source and I/O controller ID are all programmable by using this packet structure. Thus, the interrupt structure is more

flexible than those of hardwired systems.

While the invention has been shown and described with reference to particular embodiments thereof, it will be understood by those skilled in the art that the foregoing and other changes in form and detail may be made therein without departing from the scope of the invention.

Claims

1. A system for maintaining a plurality of interrupts in a data processing system having a plurality of processors, comprising:
    means for routing said at least one of said plurality of interrupts to a particular processor; and
    means for queuing said at least one routed interrupt to said particular processor.

2. A system as claimed in Claim 1 further comprising means for rejecting an interrupt if said means for routing receives a more favoured interrupt.

3. A system as claimed in Claim 1 or Claim 2 comprising means for preventing priority inversion.

4. A system as claimed in any previous claim wherein said at least one routed interrupt is queued for access by a best processor.

5. A system as claimed in any previous claim further comprising means for generating said plurality of interrupts from a plurality of interrupt sources.

6. A system as claimed in any previous claim, further comprising:
    means for accessing said at least one queued interrupt by said particular processor; and
    means for servicing said at least one accessed interrupt by said particular processor.

7. A system as claimed in Claim 6 wherein said at least one accessed interrupt is presented to said particular processor using a single source identifier.

8. A method for maintaining a plurality of interrupts in a data processing system having a plurality of processors, comprising the steps of:
    routing said at least one interrupt to a particular processor; and
    queuing said at least one routed interrupt to said particular processor.

9. A method as claimed in Claim 8 further comprising the step of generating said plurality of interrupts from a plurality of interrupt sources.

10. A method as claimed in Claim 8 or Claim 9 further comprising the step of rejecting an interrupt if a more favoured interrupt is subsequently received by said data processing system.

11. A method as claimed in any of Claims 8 to 10 wherein said at least one routed interrupt is queued for access by a best processor.

12. A method as claimed in any of Claims 8 to 11, further comprising:
    accessing said at least one queued interrupt by said particular processor; and
    servicing said at least one accessed interrupt by said particular processor.

13. A method as claimed in Claim 12 wherein said at least one accessed interrupt is presented to said particular processor using a single source identifier.

14. A system for maintaining a plurality of interrupts in a data processing system having a plurality of processors, comprising:
    means for grouping said plurality of processors into a plurality of logical servers;
    means for routing said at least one of said plurality of interrupts to a particular logical server; and
        means for queuing said at least one routed interrupt to said particular logical server.

15. A method for maintaining a plurality of interrupts in a data processing system having a plurality of processors, comprising:

15

grouping said plurality of processors into a plurality of logical servers;
routing said at least one of said plurality of interrupts to a particular logical server; and
queuing said at least one routed interrupt to said particular logical server.

5

10

15

20

25

30

35

40

45

50

55

FIG. 1

FIG. 2

FIG. 3

Interrupt Routing Layer
52

Interrupt Presentation Layer
50

Queueing and Routing Hardware

For Processor 0

Output Port → XIRR 60 (CPPR+XISR) → CPU 0 loads from BA+0 or BA+4

Input Port ← XIRR 60 (CPPR+XISR) ← Store from CPU 0 at BA+4

Input Port ← QIRR 61 (MFRR+ISSR) ← Store from any CPU at BA+12

External Interrupt Signal to CPU

72

30
IOC 1
XIVRs

71
70

30
IOC n
XIVRs

71
70

For Processor n

Output Port → XIRR 60 (CPPR+XISR) → CPU n loads from BA+0 or BA+4

Input Port ← XIRR 60 (CPPR+XISR) ← Store from CPU n at BA+4

Input Port ← QIRR 61 (MFRR+ISSR) ← Store from any CPU at BA+12

External Interrupt Signal to CPU

72

Logical Server 0 (MP Only)   38

Input Port ← G_QIRR 73 (MFRR+ISSR)   APR 39

Logical Server n (MP Only)

Input Port ← G_QIRR 73 (MFRR+ISSR)   APR 39

FIG. 4

XIRR for processor X
54

Routing Layer Selects Highest Priority 55

Queue X

53

58

59

Interrupt Routing Hardware Queue 56

MFRR Software-managed queue for 57 processor X

Hardware Queue in IOC #0
51

...

Hardware Queue in IOC #n
51

FIG. 5

IOC

ROUTING LAYER

Present interrupt start ⟶ 74

If multiple interrupts, then select one to present ⟶ 75

76 ⟶ Server # and priority from XIVR, IOID, and interrupt source # within IOC

77 ⟶ Route to specific processor ? — NO ⟶ 78 Implementation specific routing mechanism*

98 / YES

79 ⟶ Priority of int more flavored than CPPR for that proc. ?

Reject** 96 ⟵ NO

YES

80 ⟶ XISR =0? — NO ⟶ Reject** Previously accepted Interrupt or keep queued in the routing layer for later presentation 95

YES

Store IOID and source within IOC in XISR, and priority in CPPR of target processor 81

①

97 ⟶ Is specific interrupt still pending ? — YES

NO ⟶ Done with this Interrupt

Reset pending interrupt (same mechanism as reject) 94 ⟵ Hardware resets interrupt pending for interrupts queued in routing layer (if any) 93

*Implementation specific routing mechanism optimized for specific system design point. For example, optimized for cost in low end systems and for performance in high end systems.

**Reject could be to routing layer itself if XIRR is indicating an interrupt from one of the software queues (MFRR or G_MFRR not =0)

FIG. 6A

PRESENTATION LAYER

G_QIRR ⌇ 73

| G_MFRR | ISSR |

QIRR ⌇ 61

| MFRR | ISSR |

XIRR ⌇ 60

| CPPR | XISR | ← (1)
82

FIG. 6B

83 ◇ **XISR =0?** — YES

NO

84 ◇ **Interrupts enabled?** — NO

YES

85 (Begin software interrupt processing)

Software reads XIRR at BA+4, Hardware deactivates interrupt signal to processor and then: - Returns the data to the proc., - Sets the CPPR to the interrupt priority, - Sets the XISR to 0. Software saves the XIRR data   **86**

87 ◇ **XISR-> QIRR ?** → Software removes most favored queue entry and sets MFRR to new most favored entry   **88**

89 ◇ **XISR-> G_QIRR ?** → Software removes most favored queue entry and sets G_MFRR to new most favored entry   **90**

XISR -> external interrupt → Use XISR to call device driver; device driver resets interrupt at source   **91**

Write XIRR at BA+4 with value which was read from the XIRR and saved at start   **92**

FIG. 7

B'10110'

SEE TABLE 5

BUS_CLK

100 — BR

102 — BG

104 — TS

106 — ABB

108 — TT(0...4)

110 — ADDRESS

112 — AACK

114 — GBL

FIG. 8

BUS_CLK

100 — BR

102 — BG

104 — TS

106 — ABB

108 — TT(0...4)   B'10111'

110 — ADDRESS   SEE TABLE 5

112 — AACK

114 — GBL